Feuille de Travaux Pratiques sur les Matrices

```
/*****************************************************************************
        Université de Nantes - 2022/2023
        Licence Informatique
        Algorithmique et Structures de Données 1

        Travaux Pratiques sur les Matrices
*****************************************************************************/

#include<iostream>
#include<cstdlib>
#include<ctime>

using namespace std;

struct Matrice{         // Matrice de réels
        int nl;         // nombre de lignes
        int nc;         // nombre de colonnes
        double** tab2d; // tableau de réels
};

// Fonction construisant une matrice nulle de n lignes et m colonnes
Matrice zeros(int n, int m){
        Matrice M;
        int i, j;

        M.nl = n;
        M.nc = m;
        M.tab2d = new double*[M.nl];
        for (i=0; i<M.nl; i++){
                M.tab2d[i] = new double[M.nc];
        }
        for (i=0; i<M.nl; i++){
                for (j=0; j<M.nc; j++){
                        M.tab2d[i][j] = 0;
                }
        }
        return M;
}

// Procédure affectant un réel x
// au coefficient d'indices (i, j)
// d'une matrice M
// Précondition : 0 <= i < M.nl et 0 <= j < M.nc
void affecte_coeff(Matrice &M, int i, int j, double x){
        M.tab2d[i][j] = x;
}


// Fonction renvoyant la valeur du coefficient
// d'indices (i, j) d'une matrice M
// Précondition : 0 <= i < M.nl et 0 <= j < M.nc
double coeff(Matrice M, int i, int j){
        return M.tab2d[i][j];
}

// Fonction construisant la matrice identité de taille n
Matrice identite(int n){
```

```
        Matrice I;
        int i;

        I = zeros(n, n);
        for (i=0; i<n; i++){
                affecte_coeff(I, i, i, 1);
        }
        return I;
}


// Fonction construisant une matrice de dimensions (n, m)
// avec des coefficients aléatoires
Matrice hasard(int n, int m){
        Matrice R;
        int i, j;
        int random;
        double coeff;

        R = zeros(n, m);
        for (i=0; i<R.nl; i++){
                for (j=0; j<R.nc; j++){
                        random = rand() % 10;
                        coeff = (double) random/10.0;
                        R.tab2d[i][j] = coeff;
                }
        }
        return R;
}

// Procédure d'affichage d'une matrice
void affiche(Matrice M){
        int i, j;

        if (M.tab2d != nullptr){
                for (i=0; i<M.nl; i++){
                        cout << "[";
                        for (j=0; j<M.nc; j++){
                                cout << M.tab2d[i][j] << "\t";
                        }
                        cout << "]" << endl;
                }
                cout << endl;
        }
}

// Fonction permettant de calculer la somme de deux matrices A et B
// Précondition : les matrices A et B ont les mêmes dimensions
Matrice somme(Matrice A, Matrice B){
        Matrice S;
        int i, j;

        S = zeros(A.nl, A.nc);
        for (i=0; i<S.nl; i++){
                for (j=0; j<S.nc; j++){
                        S.tab2d[i][j] = A.tab2d[i][j] + B.tab2d[i][j];
                }
        }
        return S;
}
```

```
// Fonction permettant de calculer le produit d'une matrice A par un réel x
Matrice multiplication(Matrice A, double x){
        Matrice M;
        int i, j;

        M = zeros(A.nl, A.nc);
        for (i=0; i<M.nl; i++){
                for (j=0; j<M.nc; j++){
                        M.tab2d[i][j] = x*A.tab2d[i][j];
                }
        }
        return M;
}

// Fonction permettant de calculer le produit de deux matrices A et B
// Précondition : les matrices A et B ont des dimensions compatibles
Matrice produit(Matrice A, Matrice B){
        Matrice P;
        int i, j, k;
        double coeff;

        P = zeros(A.nl, B.nc);
        for (i=0; i<P.nl; i++){
                for (j=0; j<P.nc; j++){
                        coeff = 0;
                        for (k=0; k<A.nc; k++){
                                coeff = coeff + A.tab2d[i][k]*B.tab2d[k][j];
                        }
                        P.tab2d[i][j] = coeff;
                }
        }
        return P;
}

// Fonction permettant de calculer la puissance n-ième d'une matrice A
// Précondition : A est une matrice carrée
Matrice puissance(Matrice A, int n){
        Matrice M;
        int i;

        if(n==0){
                return identite(A.nl);
        }
        else {
                M = somme(A, zeros(A.nl, A.nc));
                for (i=1; i<n; i++){
                        M = produit(M, A);
                }
                return M;
        }
}

// Fonction permettant de calculer par exponentiation rapide
// la puissance n-ième d'une matrice A
// Précondition : A est une matrice carrée
Matrice expo_rapide(Matrice A, int n){
        Matrice M;

        if(n==0){
```

```
                return identite(A.nl);
        }
        else {
                if (n==1){
                        return A;
                }
                else {
                        if (n%2 == 0){
                                return expo_rapide(produit(A, A), n/2);
                        }
                        else {
                                return produit(A, expo_rapide(produit(A, A), (n-1)/2));
                        }
                }
        }
}

// Procédure de libération de mémoire
void libere(Matrice &M){
        int i, j;

        // désallocation des éléments de la deuxième dimension
        for (i=0; i<M.nl; i++){
                delete[] M.tab2d[i];
                M.tab2d[i] = nullptr;
        }
        // désallocation de la première dimension
        delete[] M.tab2d;
        M.tab2d = nullptr;
}

int main(){
        srand((unsigned) time(NULL));
        Matrice A;
        clock_t start, end ;
        int i;

        A = hasard(2, 2);
        affiche(A);

        start = clock();
        for (i=0; i<1000; i++){
                puissance(A, 1000);
        }
        end = clock();
        cout << "Puissance itérative. Temps de calcul : "
             << ( double ) (end - start) / CLOCKS_PER_SEC
             << " secondes. " << endl ;

        start = clock();
        for (i=0; i<1000; i++){
                expo_rapide(A, 1000);
        }
        end = clock();
        cout << "Exponentiation rapide. Temps de calcul : "
             << ( double ) (end - start) / CLOCKS_PER_SEC
             << " secondes. " << endl ;

        return 0;
}
```